

Euclidean TSP with few inner points in linear space

Paweł Gawrychowski¹ and Damian Rusak²

¹ Max-Planck-Institut für Informatik, Saarbrücken, Germany

² Institute of Computer Science, University of Wrocław, Poland

Abstract. Given a set of n points in the Euclidean plane, such that just k points are strictly inside the convex hull of the whole set, we want to find the shortest tour visiting every point. The fastest known algorithm for the version when k is significantly smaller than n , i.e., when there are just few inner points, works in $\mathcal{O}(k^{11\sqrt{k}}k^{1.5}n^3)$ time [Knauer and Spillner, WG 2006], but also requires space of order $k^{\Theta(\sqrt{k})}n^2$. The best linear space algorithm takes $\mathcal{O}(k!kn)$ time [Deineko, Hoffmann, Okamoto, Woeginger, Oper. Res. Lett. 34(1), 106-110]. We construct a linear space $\mathcal{O}(nk^2 + k^{\mathcal{O}(\sqrt{k})})$ time algorithm. The new insight is extending the known divide-and-conquer method based on planar separators with a matching-based argument to shrink the instance in every recursive call. This argument also shows that the problem admits a quadratic bikernel.

1 Introduction

The traveling salesman problem is one of the most natural optimization questions. Already proven to be NP-hard in the classical book by Garey and Johnson, it remains to be NP-hard even in the most natural Euclidean version [8]. A simple $\mathcal{O}(2^n n^2)$ dynamic programming can be used to solve the general version, where n is the number of points, but one can do much better by exploiting the additional properties of the Euclidean variant. This was independently observed by Smith [9], Kann [5], and Hwang, Chang, and Lee [4], who all applied a similar reasoning, which we will call the strategy of searching over separators, to achieve an $\mathcal{O}(n^{\mathcal{O}(\sqrt{n})})$ running time. Even though the problem is NP-hard, we might try to construct an algorithm whose running time depends exponentially only on some parameter k of the input instead of the whole n . We say that a problem is *fixed-parameter tractable*, if it is possible to achieve a running time of the form $\mathcal{O}(f(k)n^c)$, where k is the parameter. A closely connected notion is the one of admitting a *bikernel*, which means that we can reduce in polynomial time any its instance to an instance of a different problem, whose size is bounded by a function of k .³ In case of the Euclidean traveling salesman problem, a natural parameterization is to choose k to be the number of inner points, where a point is inner if it lies strictly inside the convex hull of the input. A result of

³ This notion is usually used for decision problem, while we will be working with an optimization question, but this is just a technicality.

Deineko, Hoffman, Okamoto and Woeginger [2] is that in such setting $\mathcal{O}(2^k k^2 n)$ time is possible (see their paper for an explanation why such parameterization is natural). This was subsequently improved to $\mathcal{O}(k^{11\sqrt{k}} k^{1.5} n^3)$ by Knauer and Spillner [6].⁴ The space consumption of their method (and the previous method) is superpolynomial, as they apply a dynamic programming on $k^{\Theta(\sqrt{k})} n^2$ states.

Contribution. Our goal is to construct an efficient linear space algorithm. As the previously mentioned exact algorithm for the non-parametrized version [4] requires polynomial space, a natural approach is to apply the same strategy. In our case we want the total running time to depend mostly on k , though, so we devise a technique of reducing the size of current instance by applying a matching-based argument, which allows us to show that the problem admits a bikernel of quadratic size. By applying the same strategy of searching over separators on the bikernel, we achieve $\mathcal{O}(nk^2 + k^{\mathcal{O}(k)})$ running time. To improve on that, we extend the strategy by using weighted planar separators, which give us a better handle on how the number of inner points decreases in the recursive calls. The final result is an $\mathcal{O}(nk^2 + k^{\mathcal{O}(\sqrt{k})})$ time linear space algorithm.

Overview. As in the previous papers, we start with the simple observation that the optimal traveling salesman tour visits the points on the convex hull in the cyclic order. In other words, we can treat subsequent points on the convex hull as the start and end points of subpaths of the whole tour that go only through the inner points. Obviously, no more than k of such potential subpaths include any inner points. We call them *important* and show that we can quickly (in polynomial time) reduce the number of pairs of subsequent points from the convex hull that can create such important subpath to k^2 , and for the remaining pairs we can fix the corresponding edge of the convex hull to be a part of the optimal tour, which shows that the problem admits a bikernel of quadratic size. The reduction shown in Section 2 is based on a simple (weighted) matching-based argument and works in $\mathcal{O}(nk^2 + k^6)$ time and linear space. The second step is to generalize the *Generalized Euclidean Traveling Salesman Problem* [4] as to use the properties of the convex hull more effectively. In Section 3 we modify the strategy of searching over separators, so that its running time depends mostly on the number of inner points. More specifically, we use the weighted planar separator theorem of Miller [7] to prove that there exists a separator whose size is proportional to the square root of the number of inner points, irrespectively of the number of outer points. Now if the number of outer points is polynomial, which can be ensured by extending the aforementioned matching-based reduction, we can iterate over all such separators. Having the separator, we guess how the solution intersects with it, and recurses on the two smaller subproblems. The separator is chosen so that the number of inner points decreases by a constant factor in each subproblem, so then assuming the reduction is performed in every recursive call, we obtain $\mathcal{O}(k^{\mathcal{O}(\sqrt{k})})$ running time in linear space.

⁴ The authors state the result for minimum weight triangulation, but the companion technical report shows that the same strategy works for our problem.

Assumptions. We work in the Real RAM model, which ignores the issue of being able to compute distances only up to some accuracy. By $d(p, q)$ we denote the Euclidean distance between p and q . In the rest of the paper, by planar graph we actually mean its fixed straight-line embedding, as the nodes will be always known points in the plane. Whenever we are talking about sets of points, we want distinct points, which can be ensured by perturbing them.

2 The reduction

We want to construct an efficient algorithm for a variant of the *Euclidean Traveling Salesman Problem*, called k -ETSP, in which we are given a set V of n points such that exactly k of them lie strictly inside $\text{CH}(V)$, which is the convex hull of the whole set. The algorithm first reduces the problem in $\mathcal{O}(nk^2 + k^6)$ time to an instance of *Generalized Euclidean Traveling Salesman Problem* of size at most $\mathcal{O}(k^2)$, and then solves the instance in $\mathcal{O}(k^{\mathcal{O}(\sqrt{k})})$ time. By the size we mean the value of $n + 2m$, where n and m are defined as below.

***Generalized Euclidean Traveling Salesman Problem* (V, T) -GETSP**

Given a set $V = \{v_1, \dots, v_n\}$ of inner points and a set $T = \{(t_1, t'_1), \dots, (t_m, t'_m)\}$ of terminal pairs of points, find a set of m paths with the smallest total length such that:

1. the i -th path is built on (t_i, t'_i) , i.e., it starts from t_i and returns to t'_i ,
2. every v_i is included in exactly one of these paths,

assuming that in any optimal solution the paths have no self-intersections, and no path intersects other path, except possibly at the ends.

It is well-known that in an optimal solution to an instance of k -ETSP the outer points are visited in order in which they appear on $\text{CH}(V)$ (otherwise the solution intersects itself and can be shortened). Hence we can reduce k -ETSP to (V', T) -GETSP by setting $V' = V \setminus \text{CH}(V)$ and $T = \{(x_1, x_2), \dots, (x_{n-k}, x_1)\}$, where $\text{CH}(V) = \langle x_1, \dots, x_{n-k} \rangle$. As any optimal solution to k -ETSP has no self-intersections, the paths in any optimal solution to the resulting instance have no self-intersections and do not intersect each other, except possibly at the ends.

We will show that given any instance of (V, T) -GETSP, we can quickly reduce the number of terminal pairs to $\mathcal{O}(n^2)$. A path in a solution to such instance is *important* if it includes at least one point from V , and *redundant* otherwise. Obviously, a redundant path consists of just one edge (t_i, t'_i) for some i , and the number of important paths in any solution is at most n . What is maybe less obvious, we can efficiently determine a set of at most n^2 terminal pairs such that the paths built on the other terminal pairs are all redundant in some optimal solution. To prove this, we will notice that every solution to an instance of (V, T) -GETSP corresponds to a matching, and apply a simple combinatorial

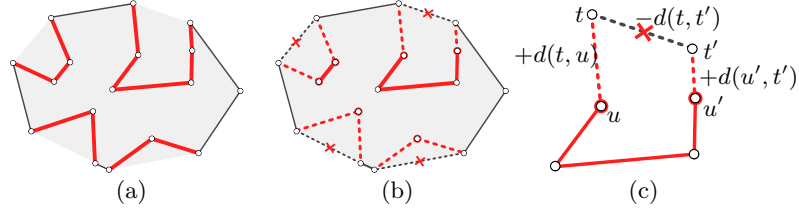


Fig. 1. (a) A solution with important paths marked with thick lines, (b) connecting the inner parts of important paths to form a solution, (c) connecting a single inner part $\langle u, \dots, u' \rangle$ to a terminal pair (t, t') costs $d(t, u) + d(u', t') - d(t, t')$.

lemma. The idea is that every important path $\langle u_0, u_1, \dots, u_\ell, u_{\ell+1} \rangle$ consists of the middle part $\langle u_1, \dots, u_\ell \rangle$ containing only inner points, and the endpoints $u_0 = t$, $u_{\ell+1} = t'$ for some terminal pair (t, t') . We create a weighted complete bipartite graph, where every possible pair of inner points (u, u') corresponds to a left vertex, and every terminal pair corresponds to a right vertex. The weight of an edge between (u, u') with (t, t') is $d(t, u) + d(u', t') - d(t, t')$, see Fig. 1(c).

First we present a simple combinatorial lemma. Given a weighted complete bipartite graph $G = (U \cup V, U \times V, c)$, where $c(u, v)$ is the weight of an edge (u, v) , $\text{cost}(X, Y)$ denotes the weight of a cheapest matching of $X \subseteq U$ to $Y \subseteq V$, if $|X| \leq |Y|$. If M is a matching of X to Y , then we denote by $M[X]$ and $M[Y]$ the subsets of X and Y matched by M .

Lemma 1. *Let $G = (U \cup V, U \times V)$ be a weighted complete bipartite graph, where $|U| \leq |V|$. If M_{\min} is a cheapest matching of U to V , then for every $A \subseteq U$ we have $\text{cost}(A, M_{\min}[V]) = \text{cost}(A, V)$.*

Proof. Assume the opposite, i.e., there is some $A \subseteq U$ such that for any cheapest matching M of A to V we have $M[V] \not\subseteq M_{\min}[V]$. Fix such A and take any cheapest matching M of A to V . If there are multiple such M , take the one with the largest $|M[V] \cap M_{\min}[V]|$. Then look at $M \oplus M_{\min}$, which is the set of edges belonging to exactly one of M and M_{\min} . It consists of node-disjoint alternating cycles and alternating paths, and the alternating paths can be of either odd or even length. Because $M[V] \not\subseteq M_{\min}[V]$, there is a vertex $x \in V$ such that $x \in M[V] \setminus M_{\min}[V]$. It is clear that there is a (nontrivial) path P starting at x , as $x \in M[V]$ but $x \notin M_{\min}[V]$. We want to argue that its length is even. Its first edge comes from M , so if the total length is odd, then the last edge comes from M as well, so the path ends at a vertex $y \in A$. But all such y are matched in M_{\min} , so P cannot end there. Hence it ends at a vertex $y \in M_{\min}[V] \setminus M[V]$, and its length is even. Now we consider three cases depending on the sign of $\text{cost}(P)$, which is the total weight of all edges in $P \cap M_{\min}$ minus the total weight of all edges in $P \cap M$:

1. if $\text{cost}(P) > 0$ then $M_{\min} \oplus P$ is cheaper than M_{\min} , so M_{\min} was not a cheapest matching of U to V ,
2. if $\text{cost}(P) < 0$ then $M \oplus P$ is cheaper than M , so M was not a cheapest matching of A to V ,

3. if $\text{cost}(P) = 0$, then $M' = M \oplus P$ is a cheapest matching of A to V , and $|M'[V] \cap M_{\min}[V]| > |M[V] \cap M_{\min}[V]|$, so M was not a cheapest matching of A to V with the largest $|M[V] \cap M_{\min}[V]|$ in case of a tie.

Hence there is a cheapest matching M of A to V such that $M[V] \subseteq M_{\min}[V]$. \square

Lemma 2. *With a read-only constant-time access to a weighted complete bipartite graph $G = (U \cup V, U \times V, c)$, where $|U| \leq |V|$, we can find a cheapest matching of U to V in $\mathcal{O}(|U|^3 + |U||V|)$ time and $\mathcal{O}(|V|)$ space.*

Proof. Let $p = |U|$ and $q = |V|$. The naive approach would be to find a cheapest matching using p iterations of Dijkstra's algorithm implemented with a Fibonacci heap [3], which uses $\mathcal{O}(p + q)$ space and $\mathcal{O}(p((p + q) \log(p + q) + pq))$ total time. We want to smaller time complexity when p is significantly smaller than q .

The first straightforward observation is that we can remove all but at most p^2 nodes from V , because we only need to keep, for every $u \in U$, its p cheapest neighbors from V . By running the aforementioned algorithm on such truncated graph, the total time becomes $\mathcal{O}(p^4)$, but the space complexity changes to $\mathcal{O}(p^2)$, which might be larger than $\mathcal{O}(p + q)$, so we need an additional idea.

We briefly recap how to use the Dijkstra's algorithm to compute a cheapest matching. We start with an empty matching and iteratively extend the current matching M using the cheapest augmenting path. An augmenting path connects an unmatched vertex $u \in U$ with an unmatched vertex $v \in V$ and alternates between the nodes of U and V . To find the cheapest augmenting path, for every $(u, v) \notin M$ we create an edge $u \rightarrow v$ with a cost of $c(u, v) - \pi_u + \pi_v$, and for every $(u, v) \in M$ we create an edge $v \rightarrow u$ with a cost of $-c(u, v) + \pi_u - \pi_v$. The *potentials* π_u and π_v are initially all equal to zero, and then maintained so that the costs of all directed edges are nonnegative, so that we can apply the Dijkstra's algorithm to find the cheapest augmenting path. Now consider a single iteration. Let E_V be the set of edges incident to the already matched vertices of U . To correctly find the cheapest augmenting path, it is enough to consider, for every $u \in U$, only the cheapest incident edge which does not belong to E_V . This reduces the complexity of a single iteration to $\mathcal{O}(p \log p + |E_V|) = \mathcal{O}(p^2)$, assuming that we can quickly extract that cheapest edge for every $u \in U$. To accelerate the extraction, for every $u \in U$ we generate a list E_u of q/p cheapest edges incident to u and not belonging to E_V . The lists are recalculated every q/p iterations. Then, in every iteration, for every $u \in U$ we know that the cheapest incident edge which does not belong to E_V belongs to the current E_u , hence it is enough to run the Dijkstra's algorithm on $|E_V + \cup_{u \in U} E_u| = \mathcal{O}(p^2 + q)$ edges, which takes $\mathcal{O}(p \log p + p^2 + q) = \mathcal{O}(p^2 + q)$ time and requires $\mathcal{O}(p + q)$ space. Because in every iteration exactly one node $v \in V$ becomes matched, recalculating the lists E_u every q/p iterations is enough.

Now we analyze how much time do we need to generate every E_u . We claim that every E_u can be found in $\mathcal{O}(q)$ time and $\mathcal{O}(q/p)$ space. We partition the sequence of all (at most) q edges incident to u and not belonging to E_V into blocks of length q/p and process the blocks one-by-one. After processing the first k blocks, we know the q/p smallest elements in the corresponding prefix

of the sequence. To process the next block, we take these q/p known smallest elements, add all elements in the current block, and use the linear time median selection algorithm [1] to select the q/p smallest elements in the resulting set of $2q/p$ numbers. After all blocks are processed, we have exactly the q/p smallest elements of the whole original sequence. The total time complexity is $\mathcal{O}(q/p)$ per every block, so $\mathcal{O}(q)$ in total, and we clearly need only $\mathcal{O}(q/p)$ space.

In every iteration we spend $\mathcal{O}(p^2 + q)$ time to run the Dijkstra's algorithm. Additionally, every q/p iterations we need $\mathcal{O}(q)$ time to recompute the lists E_u . Hence the total time is $\mathcal{O}(p^3 + pq)$. The space usage is clearly $\mathcal{O}(p + q)$. \square

Theorem 1. *Given an instance of (V, T) -GETSP with $m \geq n^2$, we can find $T_0 \subseteq T$ of size $m - n^2$, such that there is an optimal solution in which the paths built on pairs from T_0 are all redundant, in $\mathcal{O}(mn^2 + n^6)$ time and $\mathcal{O}(m)$ space.*

Proof. Let $W = V \times V$ and $G = (W \cup T, W \times T, c)$ be a weighted complete bipartite graph with W and T as the left and right vertices, respectively. The weight of an edge connecting (v, v') and (t, t') is defined as $c((v, v'), (t, t')) = d(t, v) + d(v', t') - d(t, t')$. Informally, given a path $\langle v, \dots, v' \rangle$ consisting of inner points, $c((v, v'), (t, t'))$ is the cost of replacing a redundant path $\langle t, t' \rangle$ with an important path $\langle t, v, \dots, v', t' \rangle$, assuming that we have already taken into the account the length of the inner part $\langle v, \dots, v' \rangle$, see Fig. 1(c). Now any solution corresponds to a matching in G , because for every important path $\langle t_i, v_i, \dots, v'_i, t'_i \rangle$ we can match (v_i, v'_i) to (t_i, t'_i) . More precisely, if we denote by $i_1 < \dots < i_s$ the indices of all these important paths and fix their inner parts $\langle v_{i_j}, \dots, v'_{i_j} \rangle$, then the solution corresponds to a matching of $W' = \{(v_{i_1}, v'_{i_1}), \dots, (v_{i_s}, v'_{i_s})\}$ to T , and the cost of the solution is equal to the total length of all inner parts plus $\sum_i d(t_i, t'_i)$ plus the cost of the matching. In the other direction, any matching of W' to T corresponds to a solution with the given set of inner parts (but possibly different indices of important paths). The cost of that solution is, again, equal to the total length of all inner parts plus $\sum_i d(t_i, t'_i)$ plus the cost of the matching, so any cheapest matching corresponds to an optimal solution. By Lemma 1 we know, that $\text{cost}(W', M_{\min}[T]) = \text{cost}(W', T)$, so there always is a cheapest matching of W' to T which uses only the nodes in $M_{\min}[T]$, where M_{\min} is a cheapest matching of W to T in the whole G . Therefore, we can set $T_0 = T \setminus M_{\min}[T]$, because there is at least one optimal solution, where the paths built on pairs from such T_0 are all redundant. Clearly, $|T_0| = m - n^2$. Finally, we can use Lemma 2 to find a cheapest matching, as we can implement read-only access to any $c((v, v'), (t, t'))$ without explicitly storing the graph, so the total space usage is $\mathcal{O}(n)$ and the total time complexity is $\mathcal{O}(mn^2 + n^6)$ as claimed. \square

3 Searching over separators

In this section we briefly recap the method of searching over separators used in [4] to solve the *Euclidean Traveling Salesman Problem*. At a high level, it is a divide-and-conquer algorithm. We know that an optimal solution has no self-intersections, hence we can treat it as a planar graph. Every planar graph has a

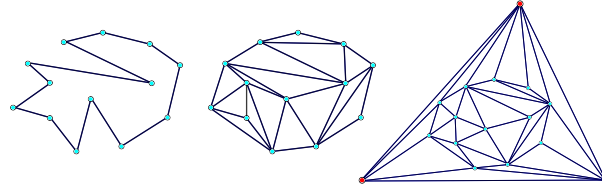


Fig. 2. A solution and its corresponding graph with $d = 9$, after adding the three enclosing points $d = 3$.

small simple cycle separator, which is a simple cycle, which can be removed as to split the whole graph into smaller pieces. Such separator can be used to divide the original problem into smaller subproblems, which are then solved recursively.

Theorem 2 (Miller [7]). *In any 2-connected planar graph with nonnegative weights summing up to 1 assigned to nodes, there exists a simple cycle, called a simple cycle separator, on at most $2\sqrt{2} \lfloor d/2 \rfloor N$ vertices, dividing the graph into the interior and the exterior part, such that the sum of the weights in each part is at most $\frac{2}{3}$, where d is the maximum face size and N is the number of nodes.*

Consider an instance of (V, T) -GETSP and its optimal solution, which by the assumption has no self-intersections, so there exists a planar graph such that any edge of the solution appears there. Then by Theorem 2 there is a simple cycle on at most $2\sqrt{2} \lfloor d/2 \rfloor (n + 2m)$ nodes such that any edge of the solution is either completely outside, completely inside, or lies on the cycle, and furthermore there are at most $\frac{2}{3}(n + 2m)$ points in either the exterior and the interior part. We want the cycle to be small, so we need to bound d . For all inner faces, this can be ensured by simply triangulating them. To ensure that the outer face is small, we add three enclosing points, see Fig. 2.

Now consider how the paths in the solution intersect with the simple cycle separator. Each path is either completely outside, completely inside, or intersects with one of the nodes of the separator. Any such intersecting path can be partitioned into shorter subpaths, such that the endpoints of the subpaths are either the endpoints of the original paths or the nodes of the separator, and every subpath is outside or inside, meaning that all of its inner nodes are completely outside or completely inside. This suggest that we can create two smaller instances of (V, T) -GETSP corresponding to the interior and the exterior part of the graph, such that the solutions of these two smaller subproblems can be merged to create the solution for the original problem, see Fig. 3.

Of course we don't know the solution, so we cannot really find a simple cycle separator in its corresponding triangulated planar graph. But the size of the separator is at most $c\sqrt{n + 2m + 3}$ for some constant c , so we can iterate over all possible simple cycles of such length, and for every such cycle check if it partitions the instance into two parts of sufficiently small sizes. The number of cycles is at most $c\sqrt{n + 2m + 3} \binom{n+2m+3}{c\sqrt{n+2m+3}} (c\sqrt{n + 2m + 3})!$, which is $\mathcal{O}((n+2m)^{\mathcal{O}(\sqrt{n+2m})})$.

Similarly, because we don't know the solution, we cannot check how it intersects with our simple cycle separator. But, again, we can iterate over all possibilities. To bound the number of possibilities, we must be a little bit more precise about what intersecting with the separator means. We create a number of new terminal pairs. Every node of the separator appears in one or two of these new terminal pairs. Additionally, the new terminal pairs might contain some of the original terminal points, under the restriction that for any original terminal pair, either none of its points are used in the new terminal pairs, or both are (and in the latter case, we remove the original terminal pair). Additionally, there cannot exist a sequence of new terminal pairs creating a cycle, i.e., $(p_1, p_2), \dots, (p_{\ell-1}, p_\ell), (p_\ell, p_1)$ with $\ell \geq 3$. Then, for every new terminal pair (p, p') , we decide if its path lies fully within the exterior or the interior part (if it directly connects two consecutive points on the cycle, we can consider it as belonging to either part). Notice that if p is one of the original terminal points, and p' is a new terminal point, then the corresponding path lies fully within the part where p belongs to. One can see that such a choice allows us to partition the original problem into two smaller subproblems, so that their optimal solutions can be merged to recover the whole solution, and that the subproblems are smaller instances of (V, T) -GETSP. Hence iterating over all choices and choosing an optimal solution in every subproblem allows us to find an optimal solution for the original instance. To bound the number of choices, the whole process can be seen as partitioning the nodes of the separator into ordered subsets, selecting two of the original terminal points for every of these subsets, and finally guessing, for every two nodes subsequent in one of the subsets, whether the path connecting them belongs to the exterior or the interior part. We must also check if it holds that for any original pair (p, p') it holds that either none of its points was selected, or both of them were, but even without this last easy check the number of possibilities is bounded by $B_{c\sqrt{n+2m+3}}(c\sqrt{n+2m+3})! \binom{2m}{2c\sqrt{n+2m+3}} 2^{c\sqrt{n+2m+3}}$, where B_s is the s -th Bell number. This is, again, $\mathcal{O}((n+2m)^{\mathcal{O}(\sqrt{n+2m})})$.

The algorithm iterates over all separators and over all possibilities of how the solution intersects with each of them. For each choice, it recurses on the resulting two smaller subproblems, and combines their solutions. Even though we cannot guarantee that all optimal solutions in these subproblems have no self-intersections, any optimal solution to the original problem has such property, so for at least one choice the subproblems will have such property, which is

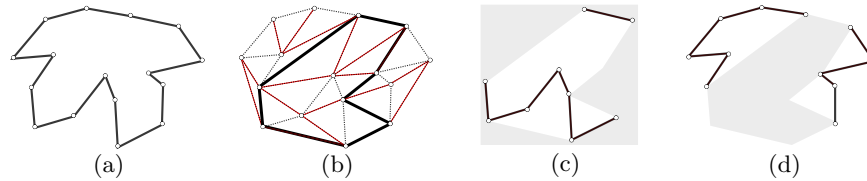


Fig. 3. (a) A solution, (b) the triangulated planar graph and its simple cycle separator, (c) a solution to the interior subproblem, (d) a solution to the exterior problem.

enough for the correctness. Because the size of every subproblem is at most $b = \frac{2}{3}(n+2m) + c\sqrt{n+2m+3}$, the recurrence for the total running time is $T(n+2m) = \mathcal{O}((n+2m)^{\mathcal{O}(\sqrt{n+2m})} \cdot 2T(b))$. For large enough $n+2m$, we have that $b \leq \frac{3}{4}(n+2m)$, and the recurrence solves to $T(n+2m) = \mathcal{O}((n+2m)^{\mathcal{O}(\sqrt{n+2m})})$. The space complexity is linear, because we only need to generate the subproblems, which requires iterating over all subsets and all partitions into ordered subsets, and this can be done in linear space.

4 (V, T, H) -GETSP

To extend the divide-and-conquer algorithm described in the previous section, we need to work with a slightly extended version of (V, T) -GETSP, which is more sensitive to the number of terminal pairs such that both points belong to the convex hull. We call the extended version (V, T, H) -GETSP, and define its size to be $n + 2m + 2\ell$. Given an instance of k -ETSP, we can reduce the problem to solving an instance of (V, T, H) -GETSP with $|V| = k$, $|T| = 0$, and $|H| = n - k$.

Generalized Euclidean Traveling Salesman Problem (V, T, H) -GETSP

Given a set $V = \{v_1, \dots, v_n\}$ of inner points, a set $T = \{(t_1, t'_1), \dots, (t_m, t'_m)\}$ of terminal pairs of points, and a set $H = \{(h_1, h'_1), \dots, (h_\ell, h'_\ell)\}$ of hull pairs of points, where for any i the point h_i and h'_i are neighbors on the convex hull of the set of all points⁵, find a set of $m + \ell$ paths with the smallest total length such that:

1. the i -th path is built on (t_i, t'_i) , for $i = 1, 2, \dots, m$,
2. the $m + i$ -th path is built on (h_i, h'_i) , for $i = 1, 2, \dots, \ell$,
3. every v_i is included in exactly one of these paths,

assuming that in any optimal solution the paths have no self-intersections, and no path intersects other path, except possibly at the ends.

We will show that if $\ell = \text{poly}(n)$, then (V, T, H) -GETSP can be solved in $\mathcal{O}((n+2m)^{\mathcal{O}(\sqrt{n+2m})})$ time and linear space using an extension of the method from the previous section. Combined with Theorem 1, this gives an $\mathcal{O}(nk^2 + k^{\mathcal{O}(\sqrt{k})})$ time and linear space solution for k -ETSP. First we extend Theorem 1.

Lemma 3. *Take an instance of (V, T, H) -GETSP with $n = |V|$, $m = |T|$, and $\ell = |H|$. If $m + \ell > n^2$ then in $\mathcal{O}((m + \ell)n^2 + n^6)$ time and $\mathcal{O}(m + \ell)$ space we can find $T_0 \subseteq T$ and $H_0 \subseteq H$ such that $|T_0| + |H_0| = m + \ell - n^2$ and there is an optimal solution in which the paths built on pairs from $T_0 \cup H_0$ are all redundant.*

Now applying the divide-and-conquer method described in the previous section directly together with the above lemma gives us a running time of $\mathcal{O}((n+2m+2\ell)^{\mathcal{O}(\sqrt{n+2m+2\ell})}) = \mathcal{O}(n^{\mathcal{O}(n)})$, and we want to improve on that to get $\mathcal{O}(n^{\mathcal{O}(\sqrt{n})})$.

⁵ Other points given in the input might or might not lie on the convex hull.

Algorithm 1 For solving (V, T, H) -GETSP.

- 1: **if** $V = \emptyset$ **then return** all edges directly connecting the pairs in $T \cup H$
 - 2: **if** $m + \ell > n^2$ **then**
 - 3: Apply Lemma 3 to find T_0 and H_0 . $\triangleright \mathcal{O}((m + \ell)n^2 + n^6)$
 - 4: Directly connect the redundant pairs in $T_0 \cup H_0$.
 - 5: Add two enclosing points I_1 and I_2 .
 - 6: **for each** ordered subset C of all points with $|C| \leq c\sqrt{n + 2m + 2}$ **do**
 - 7: Check if C forms a simple cycle.
 - 8: Check if there are at most $\frac{2}{3}(n + 2m)$ inner and terminal points in either part.
 - 9: **for each** possibility of how the solution intersects with C **do**
 - 10: Form the exterior subproblem and the interior subproblem.
 - 11: Recursively solve the exterior subproblem.
 - 12: Recursively solve the interior subproblem.
 - 13: Combine the solutions for the subproblems and update the best solution.
 - 14: **return** the best solution found in the whole process.
-

Recall that the recursive method described in the previous section iterates over simple cycle separators. Because now the (unknown) graph is on $n + 2m + 2\ell$ vertices, the best bound on the length of the separator that we could directly get from Theorem 2 is $c\sqrt{n + 2m + 2\ell + 3}$, which is too large. But say that we can show that there exists a simple cycle separator of length $\mathcal{O}(\sqrt{n + 2m + 2})$, such that the value of $n + 2m$ decreases by a constant factor in both parts. Iterating over all such simple cycle separators takes $\mathcal{O}((n + 2m + 2\ell)^{\mathcal{O}(\sqrt{n + 2m})})$ time, and iterating over all possibilities of how the separator intersects with the solution then takes $B_{\mathcal{O}(\sqrt{n + 2m})} \mathcal{O}(\sqrt{n + 2m}!) \binom{n + 2m + 2\ell}{\mathcal{O}(\sqrt{n + 2m})} 2^{\mathcal{O}(\sqrt{n + 2m})}$ time. All in all, the total number of possibilities becomes $\mathcal{O}((n + 2m + 2\ell)^{\mathcal{O}(\sqrt{n + 2m})})$, which assuming that $\ell = \text{poly}(n)$ is $\mathcal{O}((n + 2m)^{\mathcal{O}(\sqrt{n + 2m})})$. Applying this reasoning in a recursive manner as in the previous section results in Algorithm 1. Compared to the algorithm from the previous section, the changes are as follows:

1. we reduce the number of terminal and hull pairs using Lemma 3 in line 2,
2. we add just two enclosing points (instead of three) in line 5,
3. when forming the subproblems in line 10, we might connect both some terminal points and some hull points with the nodes of the separator, and in the latter case, the new pair always becomes a terminal pair in the subproblem.

If $\ell = \text{poly}(n)$ in the original instance, then we can maintain such invariant in all recursive calls without increasing the running time, because the (polynomial) cost of the reduction in a subproblem can be charged to its parent. Therefore, because the value of $n + 2m$ decreases by a constant factor in both subproblems, the total time is $\mathcal{O}((n + 2m)^{\mathcal{O}(\sqrt{n + 2m})})$ by the same recurrence as previously.

Now the goal is to prove that it is enough to consider simple cycle separators of length $c\sqrt{n + 2m + 2}$. To this end, we will prove that there exists a planar graph with the following properties:

- (a) its set of nodes includes all inner and terminal points together with the two enclosing points, and possibly some hull points,

- (b) any edge from the solution is either an edge in the graph, or lies within one of its faces,
- (c) all of its faces are of size at most 4 and its size is $\mathcal{O}(n + 2m)$.

If such a graph exists, then by Theorem 2 it has a simple cycle separator of size $\mathcal{O}(\sqrt{n + 2m})$ due to (c). Furthermore, by assigning equal weights summing up to one to all inner and terminal points, which by (a) are nodes of the graph, and zero weights to the remaining nodes, we get a simple cycle separator which, by (b), divides the original problem into two subproblems, such that the optimal solution to the subproblems can be combined to form an optimal solution to the original problem, and there are at most $\frac{2}{3}(n + 2m)$ inner and terminal points in every subproblem, so Algorithm 1 is correct. Before we show that such a graph exists, we provide the details of how to choose the enclosing points.

Lemma 4. *For any set of points A we can find two enclosing points I_1, I_2 , lying outside $\text{CH}(A)$, and two nodes of $\text{CH}(A)$ called v_{up}, v_{down} , such that:*

1. *all points of $\text{CH}(A)$ between v_{up} and v_{down} (clockwise) lie inside $\triangle I_2 v_{up} v_{down}$, and all points of $\text{CH}(A)$ between v_{down} and v_{up} lie inside $\triangle I_1 v_{up} v_{down}$,*
2. *for any point w of $\text{CH}(A)$ between v_{up} and v_{down} , $I_1 w$ has no common point with $\text{CH}(A)$ except for w , and for any w between v_{down} and v_{up} , $I_2 w$ has no common point with $\text{CH}(A)$ except for w .*

Proof. If A contains less than two points, any two I_1 and I_2 are fine. For any larger A , we can find two distinct parallel lines k_1 and k_2 , such that each of them has exactly one common point with $\text{CH}(A)$. Call these common points v_{up} and v_{down} , respectively. Let y, z' be the neighbors of v_{up} on $\text{CH}(A)$ and z, y' be neighbors of v_{down} , such that y, y' are on the other side of $v_{up} v_{down}$ than z, z' . Consider the angle β_1 obtained by extending segments $v_{up} y$ and $v_{down} y'$, and the angle β_2 obtained by extending $v_{up} z'$ and $v_{down} z$. Finally, let α_1 and α_2 be angles vertically opposite to β_1 and β_2 , respectively, see Fig. 4. Now we choose I_1 as any point strictly inside α_1 and I_2 as any point strictly inside α_2 . We must show that for such a choice both properties hold. Because of the symmetry, it is enough to prove the first part of each of them.

1. One of the properties of a convex hull is that all of its points lie inside the intersection of the halfplanes, which are defined by its segments. The intersection of the halfplanes defined by the segments $v_{up} y$ and $v_{down} y'$ is

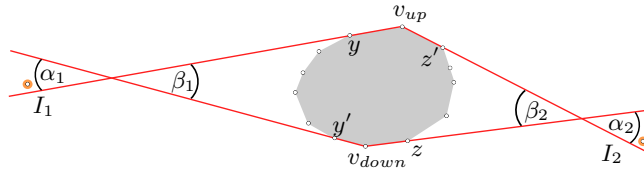


Fig. 4. Choosing the enclosing points I_1 and I_2 .

precisely β_1 . All points between v_{up} and v_{down} in the counterclockwise order lie on the same side of $v_{up}v_{down}$ as I_1 . Therefore they all lie inside the part of β_1 bounded by the segment $v_{up}v_{down}$. Due to our choice of I_1 this part lies inside $\triangle I_1 v_{up} v_{down}$, and so the first property holds.

2. Assume the opposite, i.e., the segment $I_1 w$ has a common point with $CH(A)$ other than w , call it u . Clearly, $\triangle w v_{up} v_{down}$ lies inside $\triangle u v_{up} v_{down}$ and due to the convexity of the hull all points strictly inside $\triangle u v_{up} v_{down}$ are strictly inside $CH(A)$ as well. But that is in contradiction with w being a node of $CH(A)$, and so the second property holds. \square

We say that (U, H, S) is a *hull structure* if:

1. U and H are two sets of points in the plane with $H \subseteq CH(U \cup H)$,
2. S is a collection of segments connecting the points in $U \cup H$ such that no segment intersects other segment, except possibly at the ends,
3. any point from $U \cup H$ is an endpoint of at most two segments in S ,
4. every segment in S connecting two points from H lies on $CH(U \cup H)$.

One can easily see that any optimal solution to an instance of (V, T, H) -GETSP corresponds to a hull structure (U, H, S) , where U consists of all inner and terminal points, H contains all hull points, and S is a collection of segments constituting the paths. Furthermore, for any hull structure the following holds.

Lemma 5. *If (U, H, S) is a hull structure, and I_1, I_2 are the points enclosing $U \cup H$, then there exists a planar graph, such that:*

1. *the nodes are all points from $U \cup \{I_1, I_2\}$ and possibly some points from H ,*
2. *any segment from S is either an edge of the graph, or lies within its face,*
3. *all of its faces are of size at most 4,*
4. *the size of the graph is $\mathcal{O}(|U|)$.*

Proof. The enclosing points I_1, I_2 are defined by applying Lemma 4 on $U \cup H$, same for v_{up} and v_{down} . The subsets of $U \cup H$ on the same side of the line going through $v_{up}v_{down}$ as I_1 and I_2 will be called V_1 and V_2 , respectively. The subset of S containing all segments with at least one endpoint in U will be called S' . Because any point of U is an endpoint of at most two segments, $|S'| = \mathcal{O}(|U|)$. We define U' to be the whole U together with the points of H which are an endpoint of some segment in S' , and create the first approximation of the desired planar graph using U' as its set of nodes, and S' as its set of edges. We triangulate this planar graph, so that its inner faces are of size 3. Notice that all of its edges are inside or on $CH(U')$, and all remaining segments in $S \setminus S'$ lie on $CH(U \cup H)$, see Fig. 5. So far, the size of the planar graph is $\mathcal{O}(|U|)$, its faces are small, and the nodes are all points from U and possible some points from H , and any segment from S' is an edge there. Therefore, we just need to make sure that any remaining segment is either an edge, or lies within a face.

To deal with the remaining segments, we add I_1 and I_2 to the set of nodes. Fix any point P strictly inside $CH(U')$ and, for every node P' of $CH(U')$, draw a ray starting in P and going through P' . All these rays partition the region

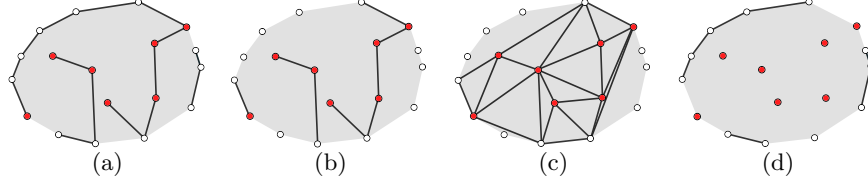


Fig. 5. (a) The segments in S , (b) the segments in S' , (c) the initial triangulated planar graph, (d) the remaining segments. Points from U are filled.

outside $\text{CH}(U')$ into convex subregions $R_1, R_2, \dots, R_{|\text{CH}(U')|}$. The intersection of R_i with $\text{CH}(U \cup H)$, called T_i , contains exactly two vertices of $\text{CH}(U')$, call them y_i and y'_i , see Fig 6(a). We will process every such T_i separately, extending the current graph by adding new triangles. Consider the sequence of points $v_{a_i}, v_{a_i+1}, \dots, v_{b_i}$ of $\text{CH}(U \cup H)$, which belong to T_i . If the sequence is empty, there is nothing to do. Otherwise we have two cases:

1. if $a_i = b_i$, create a new triangle $\triangle v_{b_i} y_i y'_i$ to \mathcal{T} , see Fig. 6(b),
2. if $a_i \neq b_i$, create two new triangles $\triangle v_{a_i} y_i y'_i$, $\triangle v_{a_i} v_{b_i} y'_i$. Then add a triangle $\triangle I_j v_{a_i} v_{b_i}$ if both v_{a_i} and v_{b_i} belong to the same V_j , see Fig. 7(a). Otherwise either v_{up} or v_{down} is in v_{a_i}, \dots, v_{b_i} , and we add three triangles as in Fig. 7(b).

Now any remaining segment which lies within a single T_i is inside one of the new triangles. To deal with the other remaining segments, for each such segment $v_j v_{j+1}$ we simply add either $\triangle I_1 v_j v_{j+1}$ or $\triangle I_2 v_j v_{j+1}$, see Fig. 7(c). This is correct because any two consecutive points on $\text{CH}(H)$ always either both belong to V_1 or both belong to V_2 . One ray can cross at most one edge, so the number of triangles created in this step is at most $|U'|$.

By the construction, the insides of any new triangles are disjoint. Also, they all lie outside $\text{CH}(U')$. Hence we can add the new triangles to the initial planar graph to form a larger planar graph. Because we created $\mathcal{O}(U')$ new triangles, the size of the new planar graph is still $\mathcal{O}(U)$, though. Now some of its faces might be large, though, so we include $I_1, I_2, v_{up}, v_{down}$ in its set of nodes, and all $I_1 v_{up}, I_1 v_{down}, I_2 v_{up}, I_2 v_{down}$ in its set of edges. Finally, we triangulate the large inner faces, if any. The size of the final graph is $\mathcal{O}(U)$ and we ensured that any segment from S is either among its edges, or lies within one of its faces. \square

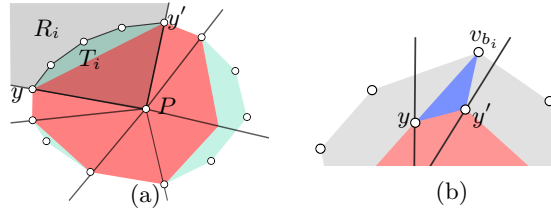


Fig. 6. (a) The intersection T_i , (b) adding one triangle when $a_i = b_i$.

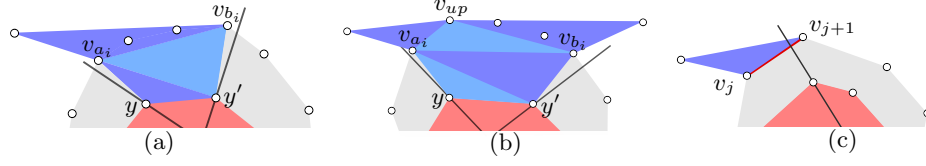


Fig. 7. (a) v_{a_i}, v_{b_i} in the same V_j , (b) v_{up} between v_{a_i} and v_{b_i} , (c) a remaining segment.

Lemma 5 shows that it is indeed enough to iterate over separators of size $c\sqrt{n+2m+2}$, hence Algorithm 1 is correct. The remaining part is to argue that it needs just linear space. By Lemma 3, the reduction in line 2 uses $\mathcal{O}(m+\ell)$ additional space which can be immediately reused. Iterating through all ordered subsets of size at most $c\sqrt{n+2m+2}$ can be easily done with $\mathcal{O}(\sqrt{n+2m})$ additional space. Bounding the space necessary to iterate over all possibilities of how the solution intersects with the separator is less obvious, but the same bound can be derived by looking at how the possibilities were counted. The $\mathcal{O}(\sqrt{n+2m})$ additional space must be stored for every recursive call. Additionally, for each call we must store its arguments V, T and H , which takes $\mathcal{O}(n+2m+2\ell)$ additional space. As $n+2m$ decreases by a constant factor in every recursive call, the recursion depth is $\mathcal{O}(\log(n+2m))$, which in turn implies $\mathcal{O}(n+2m+2\ell \log(n+2m))$ overall space consumption. Even though we always reduce the instance so that $\ell \leq n^2$, this bound might be superlinear, and we need to add one more trick.

Recall that the hull pairs in the subproblems are disjoint subsets of all hull pairs in the original problem. Hence, instead of copying the hull pairs to the subproblems, we can store them in one global array. All hull pairs in the current problem are stored in a contiguous fragment there. Before the recursive calls, we rearrange the fragment so that the hull pairs which should be processed in both subproblems are, again, stored in contiguous fragments of the global array. The rearranging can be done in linear space and constant additional space. There is one problem, though. When we return from the subproblems, the fragment containing the hull pairs might have been arbitrarily shuffled. This is a problem, because we are iterating over the ordered subsets of all points, which requires operating on their indices. Now the order of the hull pairs might change, so we cannot identify a hull point by storing the index of its pair. Nevertheless, we can maintain an invariant that all hull pairs in the current problem are lexicographically sorted. In the very beginning, we just sort the global array. Then, before we recurse on a subproblem, we make sure that its fragment is sorted. After we are done with both subproblems, we re-sort the fragment of the global array corresponding to the current problem. This doesn't increase the total running time and decreases the overall space complexity to linear.

Together with Theorem 1, this gives the final result.

Theorem 3. *k -ETSP can be solved in $\mathcal{O}(nk^2 + k^{\mathcal{O}(\sqrt{k})})$ time and linear space.*

References

1. Blum, M., Floyd, R., Pratt, V., Rivest, R., Tarjan, R.: Time bounds for selection. *Journal of Computer and System Sciences* 7, 448–461 (1972)
2. Deineko, V.G., Hoffmann, M., Okamoto, Y., Woeginger, G.J.: The traveling salesman problem with few inner points. *Operations Research Letters* 34(1), 106–110 (2006)
3. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM* 34(3), 596–615 (Jul 1987)
4. Hwang, R., Chang, R., Lee, R.: The searching over separators strategy to solve some NP-hard problems in subexponential time. *Algorithmica* 9(4), 398–423 (1993)
5. Kann, V.: On the Approximability of NP-complete Optimization Problems. *TritANA, Royal Institute of Technology, Department of Numerical Analysis and Computing Science* (1992)
6. Knauer, C., Spillner, A.: A fixed-parameter algorithm for the minimum weight triangulation problem based on small graph separators. In: *Proceedings of the 32nd international conference on Graph-Theoretic Concepts in Computer Science*. pp. 49–57. WG’06, Springer-Verlag, Berlin, Heidelberg (2006)
7. Miller, G.L.: Finding small simple cycle separators for 2-connected planar graphs. In: *Proceedings of the sixteenth annual ACM symposium on Theory of computing*. pp. 376–382. STOC ’84, ACM, New York, NY, USA (1984)
8. Papadimitriou, C.H.: The Euclidean travelling salesman problem is NP-complete. *Theoretical Computer Science* 4(3), 237 – 244 (1977)
9. Smith, W.D.: *Studies in Computational Geometry Motivated by Mesh Generation*. Ph.D. thesis, Princeton University, Princeton, NJ, USA (1989)